Online Music Search by Tapping

Geoffrey Peters, Diana Cukierman, Caroline Anthony, and Michael Schwartz

Simon Fraser University, 8888 University Drive, Burnaby, BC, V5A 1S6, Canada
{gpeters, diana, canthony, mpschwar}@sfu.ca
http://www.songtapper.com

Abstract. Query by Tapping is an emerging paradigm for content-based music retrieval, which we have explored through our web-based music search system. Based on the results obtained from our system we argue that searching for music by tapping the rhythm of a song's melody is intuitive and effective. In this paper we describe two novel algorithms to analyze tapping input. We present results indicating good accuracy rates among a broad spectrum of both trained and untrained users. Query by tapping has an important potential as a form of human-computer communication. We indicate how our algorithms to analyze tapping might be used in other areas such as music education and user authentication.

1 Introduction

One theme of Ambient Intelligence is concerned with creating ubiquitous and intuitive user interfaces to intelligent systems [1]. The act of tapping for the purpose of communication is an intuitive one, dating back to the age of Morse code and the telegraph. However, in the modern computing paradigm, rhythmic tapping is not currently a mainstream method of human-computer interaction. The current most popular methods of user input, namely the acts of "typing" and "clicking" by using the keyboard and mouse, are adequately suited for text and image based interaction, but are not very effective for content-based music information retrieval. We believe that tapping has a great potential for a rediscovery as a useful form of communication, especially as a way to express queries in music search. This so-called Query by Tapping (QBT) is an emerging paradigm for content-based music retrieval that has only just begun to be explored [23] [6] [20].

We developed an online QBT system that allows web site visitors to search for music by tapping the rhythm of song melodies. We developed two novel[1] rhythmic encoding algorithms which capture the essential elements of rhythm, allowing it to be represented in textual form. Thanks to these encoding algorithms (which we describe in Section 5), the system is tolerant of tempo variations and errors in the input. The system can also be trained by users to recognize new songs.

[1] One of these algorithms, the Rhythmic Contour String algorithm, was first presented by our team at AAAI-05, as described in [20].

Y. Cai and J. Abascal (Eds.): Ambient Intelligence in Everyday Life, LNAI 3864, pp. 178–197, 2006.

Another important advantage of our system is that it does not require any special hardware or software. The input device is the space bar on the computer keyboard, and the software is compatible with most web browser client platforms. The naturalness and accessibility of the web based interface has allowed us to gather and analyze feedback from a broad spectrum of Internet users, with varying levels of musical ability.

In Sections 2 and 3, we present an overview of previous work in tapping-based music retrieval, and a general background relating QBT to other forms of Music Information Retrieval (MIR). In Section 4 we describe the design and implementation of our web-based QBT system, including an automatic learning module allowing users to train the system. In Section 5 we then discuss the mechanics of two novel algorithms for analyzing tapping. In Section 6 we analyze user feedback and usage data collected from our web-based QBT system to provide some promising results concerning the effectiveness of the system. Based on our experimentation and analysis of the data obtained, we conclude that tapping as a form of human-computer communication has an important potential for various future applications.

2 Tapping as a Natural Act

An underlying theme behind Ambient Intelligence is that computers and humans should interact in a way that is unobtrusive and natural [1]. This involves an intuitive user interface as well as enabling the computer system to have enough information available to make appropriate decisions in order to communicate effectively with the user.

Across all cultures, people move their bodies to the rhythms of music, by clapping, tapping, drumming, singing, dancing, or rocking an infant [22]. As we have discovered through a study of user experiences with our QBT system, the act of tapping to express a musical rhythm is surprisingly easy to many people, even if they are not musically trained. While pitch-based approaches such as humming or whistling have potentially more descriptive capability for song melodies, they are not always as intuitive as tapping, nor are they as accurate for musically untrained or tone deaf users.

The concept of clapping along to a song is very natural. Many children learn to play clapping games such as patty-cake. At music concerts, audiences sometimes spontaneously clap in a specific rhythm, along with the musicians on stage. In addition to clapping their hands, people commonly use other physical motions to follow rhythms, such as foot stomping, knee smacking and finger snapping.

The task we propose, through our QBT system, is for users to tap the rhythm of a song's melody on their computer keyboard's space bar, in order to retrieve a list of songs which have melodies that contain a similar rhythm. Because of the underlying rhythmic encoding algorithms our system employs, users tap at their own tempo and pace. Overall tempo variations such as speeding up or slowing down do not produce adverse effects. Users may begin and end tapping anywhere within the song, and errors in the input are tolerated.

Other QBT systems, which were developed independently, use different input methods (as well as different underlying algorithms). Jyh-Shing et al's system [23] allows the user to tap a rhythm on the end of a Karaoke microphone, and uses a matching algorithm based on timing vectors and dynamic programming. Eisenberg et al's system [6] requires the user to tap at a certain tempo on a specialized drum pad (using either the hands or drumsticks), and uses an algorithm based on MPEG-7 beat vectors. Jyh-Shing and Eisenberg both provide evidence to support the claim that beat information is an effective feature for song search in a music database. The QBT system that we developed extends the tapping human-computer interface concept further, in terms of ease-of-use and accessibility, by being the first QBT system to be deployed as a web application. Our encoding and matching algorithms produce comparable, if not superior performance to existing systems, and also have the advantage of relative simplicity and ease-of-implementation.

The rhythm of a song's melody is often equivalent to the rhythm of the words in the lyrics, as the song is commonly sung. Each syllable or two of a word might represent one beat. Whether or not the user remembers the specific words, he or she can often recall and reproduce (to some extent) the rhythm and most likely the tone of the melody. We have found from user observation that tapping the melody of a song is quite easy to do, if the user sings aloud and taps at the same time.

3 Overview of Music Information Retrieval

Music Information Retrieval (MIR) is a field that is concerned with the general problem of searching for music in a library based on the content of the music, rather than on textual meta-data such as the song title or artist name. Users may express queries based on features obtained from existing recordings (such as in Query by Example), or through inputting a performance of the rhythm or pitch of a song's melody (such as in Query by Tapping or Humming respectively).

The motivation behind research in MIR extends to both academic and commercial applications. Melucci et al [17] emphasize that music is an important form of cultural expression, and with increasing digital access, librarians need more effective methods for organizing and retrieving it. Kosugi et al [13] describe a Karaoke machine that lets users select the song they want by singing a part of it.

Query by Example, as previously mentioned, describes the situation where a user would provide the system with a music recording, and the system would extract features from the recording to allow the discovery of other music with similar features. Examples of such as systems are discussed in [10] and [11].

In contrast with Query by Example, another area of MIR is based on user-input, where users search for a song by some element of its musical content, using a natural input method such as tapping, humming, or singing. The overriding focus in user-input based MIR is on the melody of songs, since that is the musical component most easily identifiable by musically untrained users [17]. In addition to QBT, which utilizes the rhythmic features of a melody, other input methods

are Query by Humming or Singing, which utilize the pitch features of a melody [8] [14] [12]. There are also systems which incorporate both rhythm and pitch [5].

Some user-input based MIR systems rely on extended knowledge from the user, such as musical training or the ability to read music [6]. Ideally, the user interface to a MIR system should be so intuitive that the user does not need special training. As we further analyze in Section 6, some users of our QBT system did not have any musical experience, and were still able to achieve reasonable success rates.

Another common feature of MIR systems is a tolerance to various types of user input errors. This is especially important if users have varying levels of musical ability. The way errors are handled depends on the algorithm which is used to analyze the songs and find matches. Techniques such as dynamic programming, n-grams, and approximate string matching have been used in previous work [26]. Kline & Glinert [12], as well as Kosugi et al [13] focused on trying to improve robustness against specific types of user errors.

In our QBT system, by focusing on the rhythmic features and ignoring pitch features, we need not worry about errors in pitch. Some users could be tone-deaf, or just incapable of playing the song with the correct pitch. For dealing with errors in rhythm, our system employs a fast approximate matching algorithm which we describe in Section 5.1.

Scalability of MIR systems is another important feature. As Bainbridge et al [2] describe, accuracy in matching is not the only scalability concern; we also should worry about the run-time complexity of the algorithm in general. They claim existing algorithms based on approximate matching are too slow and impractical for databases larger than 10,000 songs, when a linear scan of the database is used. They describe a Bioinformatics heuristic approach called BLAST, which provides a database indexing scheme that they purport can do much better. In order for this, and other general scalability questions, to be answered, further empirical testing is needed.

A key challenge with many MIR systems is the task of creating a song database that has an appropriate music representation format. Many MIR systems represent melody and rhythm in secondary formats, which cannot be easily determined from digital audio recordings that are commonly found in music libraries. User-input based approaches rely on higher-level music representation forms such as MIDI (Musical Instrument Digital Interface) which specify precise note and timing information for the melody, instead of working directly on a digital audio signal. Although attempts at pitch and rhythm extraction from audio recordings have been made [9], the current state of the art cannot accurately extract melodies and rhythms from the majority of audio recordings. Unfortunately, this means that databases containing higher-level music representations need to be created manually, or through techniques such as Optical Music Recognition [2].

Our web-based QBT system attempts to overcome the lack of available song data by allowing web visitors to train the system by tapping new songs, and then associating the tapping data with specific song names. This user-driven training approach presents some challenges of its own, but is part of a growing number

of web-based Artificial Intelligence systems that take advantage of the "collective mind" of Internet users [3] [4] [28]. We describe this user-driven training mechanism in Section 4.3.

Web-based QBT systems (such as ours) can be deployed using the hardware and software that already exists on the majority of Internet-enabled workstations. Without any special hardware, our QBT system allows users to 'tap' using the space bar on their computer keyboard. Cross-platform compatibility is achieved by software deployment using web standards such as HTML, Java Applets, and Macromedia Flash, which can be accessed from most modern browsers such as Firefox or Internet Explorer. Our web-based QBT system provides a MIR system that is nearly universally accessible, through its intuitive interface and cross-platform compatibility.

4 Our Application: Web-Based Query by Tapping

We developed a web based system for music search by tapping, using the algorithms described in Section 5. The initial prototype system allowed searching of a database of 30 children's songs, and collected feedback from the users about the success of their tapping experience and search results. The subsequent second-generation system allows users to expand the song database by training the system themselves. In both systems, the user taps the rhythm of a song's melody on his/her computer keyboard and the tapping sequence is recorded using a Java applet (or Macromedia Flash applet, as in the second-generation system) and then is sent to our application server for analysis. The search results are displayed in the browser, and the user has the opportunity to give feedback on the outcome of the search.

By using existing, commonplace web-based standards and technologies for application delivery, as well as the ubiquitous standard computer keyboard for user input, we have allowed a broad audience of Internet users to access our system, who may not be experts in either music or computer technology. Access to this larger audience, provided with our online feedback system, has allowed us to gain new insights into the applicability and usefulness of a QBT system on a larger scale than a controlled laboratory environment.

4.1 User Interface

A user can visit our web site [21], and use a Java or Flash applet to tap a rhythm of a song, using the space bar on the keyboard (see Figure 1). The applet will generate a MIDI file and automatically upload it to our application server, which will display the search results in the browser. This allows users to input rhythmic data without having special equipment such as a MIDI keyboard or a microphone device. The system relies on users having sufficient musical skill to input a rhythm that can be accurately discriminated. As we describe in Section 6 we have found that many untrained users do possess enough rhythmic skill to use the system. By using a web based interface, users can be located in diverse parts of the globe, and still interact with our software.

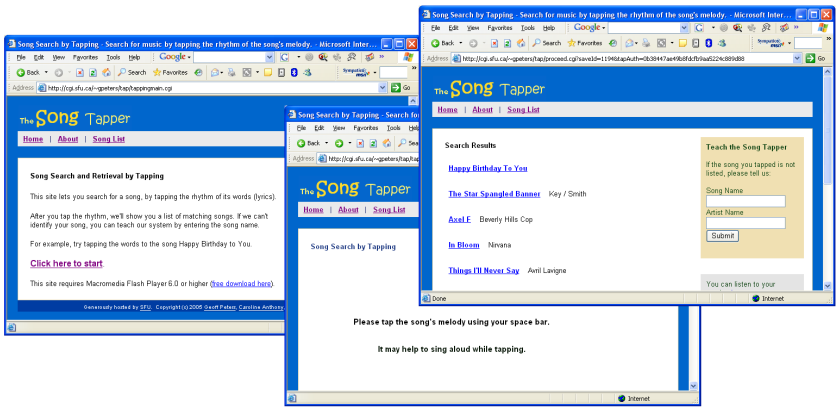

Fig. 1. Screenshot of Online Song Search by Tapping

4.2 Architecture

Our software makes use of the MIDI (Musical Instruments Digital Interface) standard to represent recordings of user tapping sessions. A musical performance is encoded as a series of events, with each event having properties such as the time, duration, and velocity of the note. When a user begins tapping, a new event would be recorded for each time the space bar is pressed on the keyboard. A special MIDI keyboard is not required for our tapping application, as we have utilized the standard alphanumeric computer keyboard as a tapping device. Our client-side browser applet, originally implemented in Java and now also implemented in Flash, allows a MIDI file to be generated from a tapping performance on a computer keyboard. On the server side, our Perl-based software makes use of the Perl::Midi library to parse and analyze MIDI files, which are passed through our encoding algorithms (as described in Section 5) to generate strings that represent the rhythm. The Perl String::Approx library is used to carry out fast approximate string matching. In addition, a MySQL database is used to store the song data.

4.3 Web Based Training Module

One problem, as previously identified, is the lack of a large database of songs which can be used as training data for our system. We are currently overcoming this difficulty by allowing the visitors of the site to train our system in real time. Our database of songs grows incrementally thanks to training performed by volunteer visitors.

Here is a listing of steps in the process that this training occurs:

1. *User Input of Song Rhythms*
   To search, the user taps the rhythm of a song using our web site tapping applet (by tapping the space bar on the computer keyboard). A MIDI file is generated automatically, and the song is uploaded to the application server.

2. *Data Processing, Search, and User Feedback*
   If the system guesses the song correctly then the user gives positive feedback indicating that this is the case, and the user's data is added to candidate rhythms for that song.
   When subsequent searches are done the software searches through all candidate rhythms for all songs for a more accurate search.
   If the system does not guess correctly then the name of the song is entered by the user (if known), and new training data is added to the database. To facilitate more accurate training, the system shows a list of songs with similar names that are already in the database (by doing a metadata search), allowing the user to add the data to an existing song or create a new song.

Some users naturally have more skill at training the system than others, due to their musical ability. In the near future, a feature will be added whereby regular users will be able to sign in, and if they generally input more accurate rhythms then their input will be given more weight when the search is done.

The data in the database which matched incorrectly (when searching) is tracked and if it accumulates too many incorrect matches then it is flagged as possible bad data. The system also tracks good matches.

This training module, which we have recently implemented and launched on our web site, has already begun to increase the number of songs in our song database. As the database grows, it will become more useful for users, hopefully attracting more users to train the system. We expect that we will reach a point where we will be able to correctly identify a good portion of the rhythms that are tapped into the system. Once the database has reached such a useful state, it should be possible to connect our system to an online music store, or even make it accessible on mobile devices to take advantage of the growing mobile/wireless music market.

5 Our Algorithms for Query by Tapping

We present two algorithms, the Rhythmic Contour String algorithm and the Phrase String algorithm. These algorithms encode rhythm in textual string form, and specify how these strings should be compared using approximate string matching to determine rhythmic similarity.

Our algorithms are designed to analyze a monophonic tapping sequence (that is, a sequence in which only one key is ever depressed at any one time). When a tapping session is performed by a user, we record the time at which the user depresses and releases the key for each tap. These key presses are represented in a sequence of note onset times o_i and release times r_i. In our analysis, we are only concerned with the timing of when the user strikes the keyboard for each tap. We are not at all concerned with the timing of when the user releases each key, assuming the user must release the key in order to strike the next tap. The exception is the last note in the sequence, for which we do consider the release time because there is no subsequent note to follow.

In this discussion, we consider a beat's duration to consist of the time between the onset of the current tap to the onset of the next tap. An entire tapping session, then, consists of a sequence of n beat durations d_i where $d_i = o_{i+1} - o_i$, except for the last d_n which takes the value of $d_n = r_n - o_n$.

In order to allow for global tempo independence, we normalize the durations of the beats. To normalize the durations, the average duration of a beat is calculated, and then each beat's duration is divided by the average duration. For a particular song that is being analyzed, the graph of the normalized duration can be plotted per beat, as beats progress through time. By comparing these "duration plots" for various songs, it can be observed that most songs have unique "duration functions". Figure 2 shows the duration plot for a user's performance of the first part of the song 'Are You Sleeping'. The corresponding sheet music notation is shown in Figure 3.

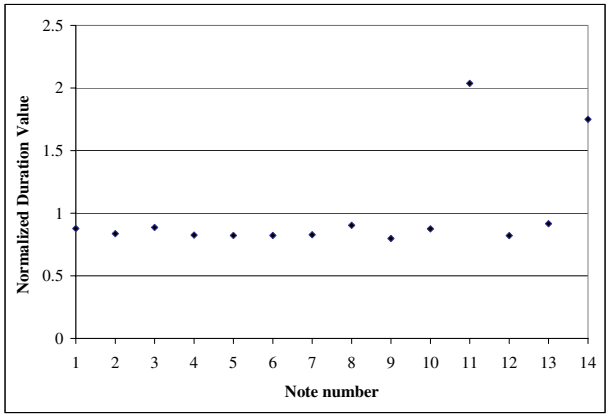

Fig. 2. Normalized Duration Plot of 'Are You Sleeping'

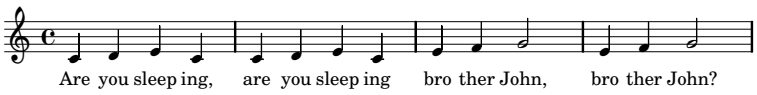

Fig. 3. Sheet Music Notation for 'Are You Sleeping'

5.1 Rhythmic Contour String Algorithm

Our goal is to encode the tapping duration information d_i as a string of characters, so that approximate string matching can be used, leveraging fast existing algorithms (such as Sun and Manber's algorithm [25]). To achieve the string encoding, we introduce the idea of a Rhythmic Contour. We conceive the idea of a rhythmic contour as a description of how the duration of each consecutive beat is different from the previous one. The rhythmic contour is calculated by

Algorithm 1. Encode Rhythmic Contour String *str* from n Beat Durations d_i

Require: d_i is an array of normalized beat duration values where i takes values from 1 to n. T is a similarity threshold value > 0 such as 0.25.
Ensure: *str* is a string of length $n-1$ consisting of any of the characters s, u, and d.

```
str ← empty string
for x = 1 to n − 1 do
    c ← d_{x+1} − d_x
    if |c| < T then
        append character 's' to string str
    else if c < 0 then
        append character 'd' to string str
    else
        append character 'u' to string str
    end if
end for
return str
```

subtracting the duration of each pair of consecutive notes. Thus we calculate the contour values c_i as $c_i = d_{i+1} - d_i$.

The result of our Rhythmic Contour String encoding for a song of $n+1$ beats is a string with n characters, with the three symbols 's', 'd', and 'u' (meaning the duration stays the same, goes down, or up, respectively). Our encoding algorithm produces a character for each sequential contour value c_i. For each contour value c_i, if $|c_i| < T$ (where T is the value of a threshold such as 0.25), we append the character 's' to the string. Failing that, if $c_i < 0$ we append the character 'd' to the string, or if $c_i > 0$ we append the character 'u' to the string.

For example, a song that has n beats of equal value would have n-1 calculated contour values, each with a value of 0, and therefore a string of $n-1$'s' characters. A simple song that has only two beats, the last with a much longer duration than the first, would have a single, positive contour value, and a string with simply the character 'u'. It may be illustrative to examine a plot of the Rhythmic Contour values (see Figure 4) and associated Rhythmic Contour String ('sssssssssudsu') for the example song, and compare it to the plot of duration for the same song (see Figure 2).

A related approach to analyzing rhythm was presented by Doraisamy and Ruger [5], where ratios between note onset times were examined. They observed that by comparing the differences between consecutive note onset times, exact beat and measure information does not need to be known, and quantization on a predetermined base duration is not required. They also remark that the usage of note onset times was earlier explored by Shmulevich et. al [24].

Approximate Matching Procedure. Each song to be put in the database is pre-processed by the above algorithm and a Rhythmic Contour String is generated for each. When a query is received from the user, the user's input rhythm is analyzed, and a Rhythmic Contour String is generated for it as well. Now that both the user's input rhythm and the rhythms of the songs in the database

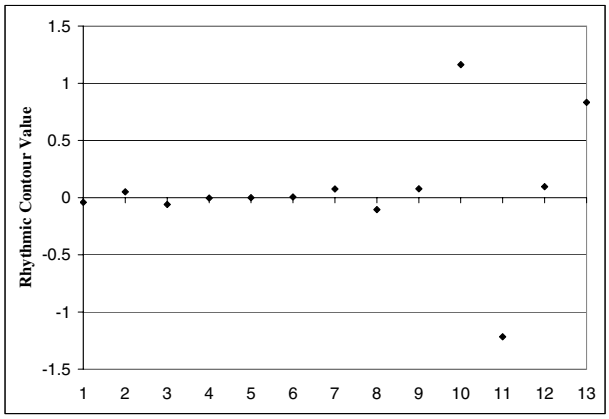

Fig. 4. Rhythmic Contour Plot of 'Are You Sleeping'. These rhythmic contour values would produce the string 'sssssssssudsu'.

are represented as strings, we use an approximate string matching procedure to determine the closest matching songs. A fast approximate string matching algorithm as specified by Sun and Manber [25] is used to calculate the edit distance between the input string and each string in the database[2].

The edit distance (also known as the Levenshtein measure [15]) is defined as the minimum number of transformation operations needed to transform one string into another string. Transformations can be composed of one or more of the following three operations:

1. removing a character
2. inserting a character
3. substituting one character for another

The edit distances between the input string and each string in the database are sorted into ascending order. The song's string in the database with the least edit distance to the input rhythm string is considered to be most probable match.

Similarity to Parsons Code. Our Rhythmic Contour String encoding is similar to a code developed by Denys Parsons in 1975, called the Parsons Code [19]. Our approach uses a string to describe the contour of how a melody's *rhythm* changes through time, whereas the Parsons Code describes the contour of how a melody's *pitch* changes through time. Parsons created a dictionary of musical themes, where he used the melodic contour of a song (the way in which a melody goes up or down on the music scale/staff) to encode a string with the letters 'U', 'D' or 'R' for up, down, and repeat, respectively. For example, the Parsons Code for the beginning of "Twinkle Twinkle Little Star" is RURURDDRDR-DRD. Our Rhythmic Contour String encoding is essentially a Parsons Code for rhythm.

[2] A freely available Perl implementation of this algorithm can be found in the Perl String::Approx library, at `http://search.cpan.org/`

Some researchers have used the Parsons Code as a basis for a MIR system, such as Tseng's implementation [26] which uses melodic (pitch-based) contours. Another MIR application which uses Parsons Code can be accessed at Musicpedia [18]. The Musicpedia application allows the user to input the Parsons Code directly, or to hum into a microphone to generate the Parsons Code. For some users, direct input of a Parsons Code or Rhythmic Contour String may seem unnatural or unintuitive. Thus, our system does not require the user to input a Rhythmic Contour String directly, but rather generates the appropriate string based on the user's tapping performance.

Scalability of the Algorithm. The string encoding procedure in Algorithm 1 does not pose a scalability concern since the number of beats is relatively small, and the number of operations is linear to the number of beats. Songs that are added to the database only need to be encoded once, and the encoded strings are stored for future use.

For the approximate string matching algorithm, as described by Sun and Manber [25], the matching of the encoded query string with a single song in the database takes $O(nk\lceil m/w\rceil)$ where n is the size of the string in the database, k is the number of errors allowed in the input, m is the length of the encoded query string, and w is the system word size (for example: 32 bits). Interestingly, because of the dependence on system word size in the matching algorithm, on a system with a 32 bit word size, queries that have between 34 to 65 taps take twice as long as queries that have up to 33 taps. Queries with over 65 taps take at least three times as long. That is because a session with $m+1$ taps generates a contour string with m characters; thus, for example, a query with 34 taps generates a string with 33 characters which exceeds the system word size of 32.

This approximate string matching for a query in our system is currently performed for each song in the database, in a linear scan. Thus the search time grows linearly with the number of songs in the database. As Bainbridge et al [2] describe, such a linear scan poses a scalability concern when the database is on the order of 10,000 songs. Some possible ways to allow the database to scale further would be to use a parallel server configuration to divide up the load of the query on to multiple servers, or to develop a partial indexing scheme (borrowing ideas from the BLAST Bioinformatics system mentioned by Bainbridge et al), which would reduce the number of songs that would need to be examined for a particular query. As previously mentioned, future work needs to be done to address these scalability concerns.

5.2 Phrase String Algorithm

We have also developed a Phrase String algorithm to be used in conjunction with the Rhythmic Contour String algorithm. We first sort the song matches by the edit distance obtained by the Rhythmic Contour String algorithm, and then further sort by the edit distance obtained by the Phrase String algorithm. Using these two algorithms in conjunction appears to produce even more accurate results than using a single algorithm alone. Further empirical testing is currently in progress.

Algorithm 2. Encode Phrase String *str* from n Beat Durations d_i

Require: d_i is an array of normalized beat duration values where i takes values from 1 to n. $n \geq 1$.
Ensure: *str* is a string consisting of the characters '1' and 'p'.

```
1: str ← empty string
2: for x = 1 to n do
3:    append character '1' to string str
4:    if IsEndOfPhrase(d_x) then
5:       append character 'p' to string str
6:    end if
7: end for
8: return str
```

The Phrase String algorithm is similar to the Rhythmic Contour String algorithm in that both algorithms encode a string based on a sequence of duration values d_i and then use approximate string matching to determine the edit distance between a query string and strings in the database.

The Phrase String algorithm generates a "phrase string" which contains the symbol '1' for each beat in the song. Additionally, for each beat which denotes an end of a musical phrase, the symbol 'p' is inserted after the '1' for that beat. The phrase detection algorithm which we have implemented identifies beats with durations that are longer than the two closest neighboring beats, as possible ends of phrases. This phrase detection approach is a much simplified version of the algorithm described by Melucci and Orio [17]. For example, the Phrase String for the duration values shown in Figure 2 would be '11111111111p111p'. In this example, there are 14 characters of '1', since there are 14 beats. A 'p' is inserted after the eleventh '1' because that beat is longer in duration than the two neighboring beats. A 'p' is always added at the end of the string, because we expect that the user would likely stop tapping at the end of a phrase.

Approximate string matching can again be used on this Phrase String, with a slight modification to the approximate string matching algorithm. Instead of allowing three different transformation operations to calculate the edit distance, only the following two are allowed:

1. removing a character
2. inserting a character

The transformation operation that is not directly allowed is the substitution of a character for another. The reason for this restriction is that the symbol '1' denotes a beat, and 'p' denotes an end-of-phrase, and these concepts are not obviously substitutable. In the Rhythmic Contour String algorithm we do allow substitutions because each character represents a transition from one beat to another, and a substitution simply implies that one of the notes had an incorrect duration. But for the Phrase String, a substitution implies that the user made an error where he/she intended to play a longer note, but instead

played an extra note, or vice versa. We consider this to be an error worth an edit distance penalty of 2 instead of 1 (that is, it is accounted for by an insertion and a deletion). Any mismatches between the strings will be detected by the two allowed transformation operations. Thus this algorithm still allows for errors such as the insertion of an extra beat, or a missing end-of-phrase symbol.

6 Results and Analysis

Our initial prototype system, for which the results will be analyzed in this section, made use of the Rhythmic Contour String algorithm as described above, and had a database of 30 children's songs. It did not make use of the Phrase String algorithm, which was developed after the initial prototype was created. However, the Phrase String algorithm is implemented in our second generation system, which is currently collecting data from users who access it.

For the initial prototype system, users were asked to provide feedback on whether the song they tapped was identified as the first song in the search results, or failing that, if the song was in the top ten results. Users were also asked to self-rate their musical ability on a scale of 1 to 5. The users' tapping sessions were recorded and data was collected such as the duration of each session, and the number of notes tapped.

Users gave feedback for 518 tapping sessions, performed from May to September 2005. Regardless of user ability level, 62% of all feedback reports indicate a first place ranking, that the system determined the correct song as the first ranked search result. 20% of all feedback reports indicated that the correct song was in the top ten, but was not the first ranked result. 18% of all feedback reports indicate the correct song was not present in the top 10 search results.

Figure 5 shows the breakdown of the self-rated ability levels of the users in the collected data. Figure 6 shows that the proportion of first place ranking outcomes is largest for experts, with a proportion of 73.8%. Beginners have the lowest first place ranking outcome proportion, of 39.7%. This data suggests that musical ability of users plays a significant role in the overall usability and effectiveness of the system, although beginners can still experience success. Surprisingly, those who rated themselves as having 'no musical experience' did better than the beginners, with a first place ranking proportion of 55.8%.

Figure 7 relates the accuracy rate to the number of notes tapped in the session. The accuracy rate is calculated as the proportion of the sessions which had a first place ranking outcome. One can observe that sessions with fewer than 10 notes tapped had an accuracy rate of zero, and from this point onwards, the accuracy rate appears to increase linearly with number of notes tapped, until 25 notes are tapped. After 25 notes tapped and onward, the accuracy rate appears to remain fairly constant. The oscillation observed from 50 notes and above is likely due to a lower concentration of data with these number of notes. The range of this graph encompasses 98.26% of the feedback data, as the number of sessions with 60 notes tapped and greater were not numerous enough to provide an interesting graph beyond that point.

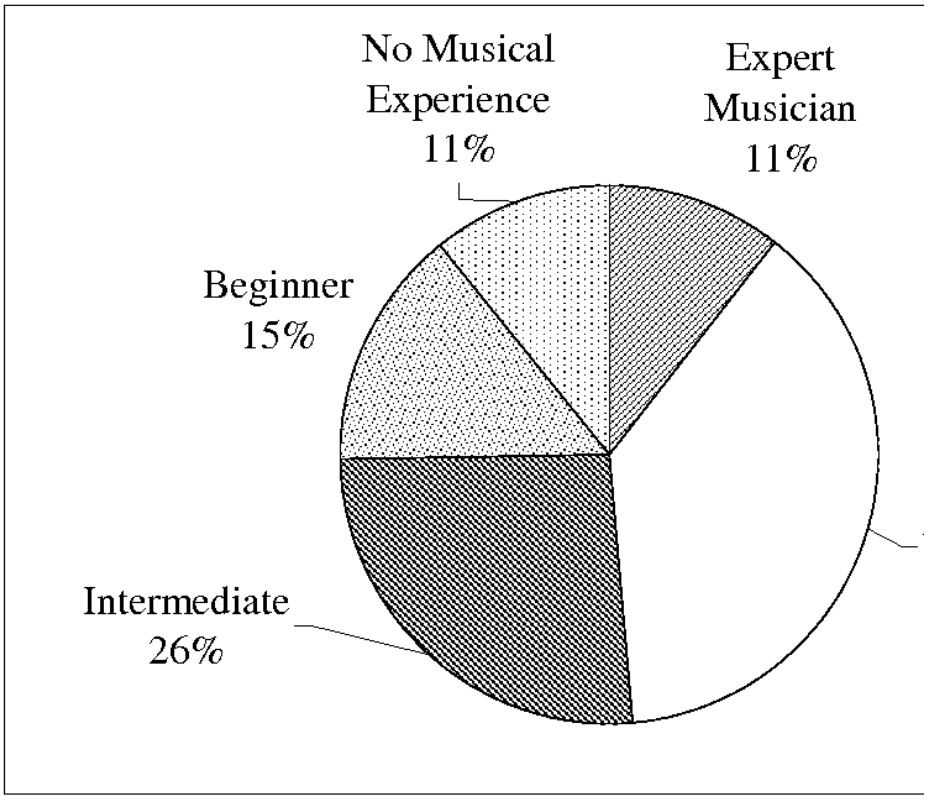

Fig. 5. Self-Reported Ability Levels of Users. For each of the 518 tapping sessions, users were asked to self-rate their musical ability level.

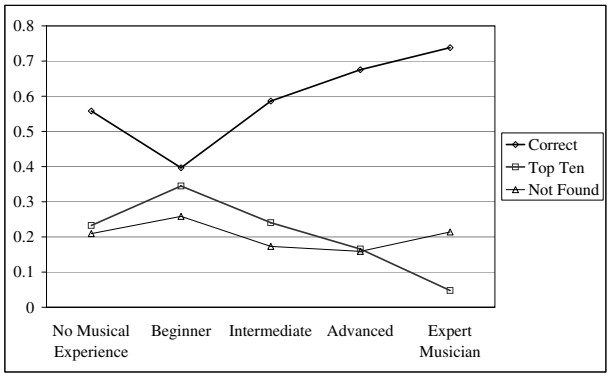

Fig. 6. Breakdown of Outcome by Ability. The vertical axis shows the proportion of total searches that had the specified outcome, for the particular user ability level.

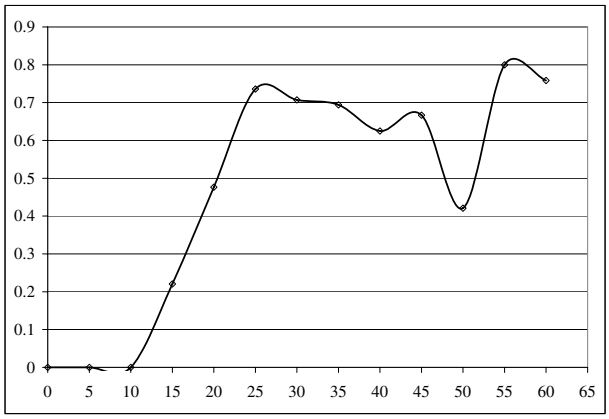

Fig. 7. Accuracy Rate Versus Number of Notes Tapped

Figure 8 relates the accuracy rate to the length of the session in seconds. As before, the accuracy rate is calculated as the proportion of the sessions which had a first place ranking outcome. A positive correlation is apparent until the session length reaches 8 seconds. From this point onwards, the accuracy rate appears to remain fairly constant. The range of this graph encompasses 90.7% of the feedback data, as the number of sessions with length 20 seconds or greater were not numerous enough to provide an interesting graph beyond that point.

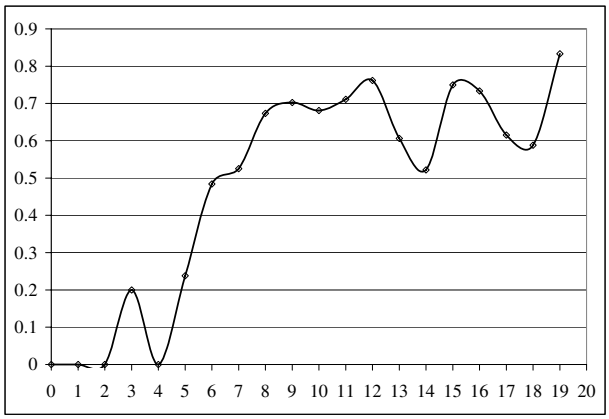

Fig. 8. Accuracy Rate Versus Session Length in Seconds

When accessing our system, users were not given any instructions as to how many taps they should make, or how long their tapping session should take. Users were free to tap at their own tempo, and were not restricted with a metronome. The distribution of sessions by number of taps is shown in Figure 9 and has a

mean of 30.8 taps and a standard deviation of 15.2. The distribution of sessions by length of session in seconds is shown in Figure 10 and has a mean of 11.8 seconds and a standard deviation of 6.75. We observed that on average, users tend to tap enough notes to provide the system with adequate information for a successful search. Notice that users enter an average of 30.8 taps (Figure 9) and that the system performs the best when at least 25 taps are entered (Figure 7). A similar observation can be made connecting results in Figure 10 and Figure 8.

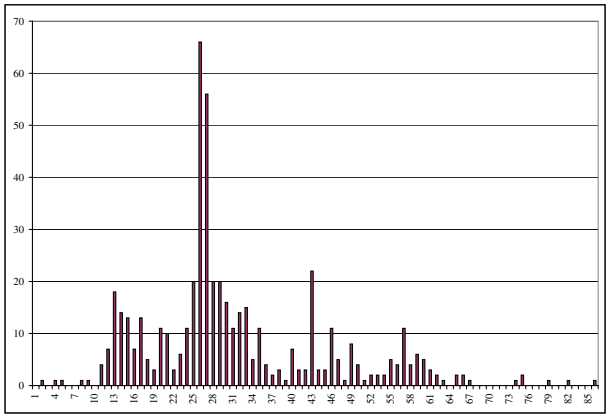

Fig. 9. Histogram of Sessions by Number of Taps. The mean number of taps is 30.8 and the standard deviation 15.2.

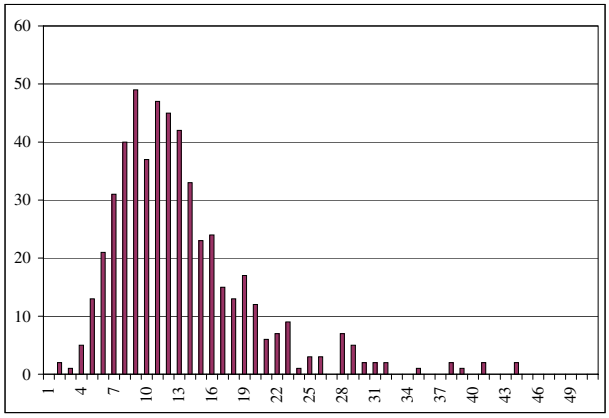

Fig. 10. Histogram of Sessions by Length of Session. The mean session length in seconds is 11.8 and the standard deviation 6.75.

It is expected that if the database grows to several thousand songs, the Rhythmic Contour String algorithm alone will no longer provide enough information to discriminate song content accurately. However, the combination

of the Rhythmic Contour String algorithm with the other algorithms (such as the Phrase String algorithm described above), will likely increase the discriminatory ability of the system.

7 Other Applications of Query By Tapping

Computer-aided music education is one possible application area of our tapping algorithms, which we believe merits future exploration. We conducted an informal experiment at AAAI-05 [20] where about 50 subjects attempted to tap a song using our system, and those who had difficulty in tapping the correct rhythm were asked to observe an expert musician tapping the same song. By learning through imitation, the subjects were able to improve the accuracy of their tapping on a subsequent attempt, as evidenced by a smaller edit distance to the intended song. In this experiment, rather than using the tapping interface as a search tool, it was instead used as a way to assess the subject's musical knowledge of a particular song. Such a tool could be useful in music classrooms to provide immediate feedback to students who are learning new rhythms.

Some other application areas, which are inspired by the themes of Ambient Intelligence, are to integrate tapping interfaces into mobile and embedded applications. For example, if a tapping sensor were integrated into the home, a user could simply tap a certain rhythm on a table-top to cause the lights to dim, and a romantic song to begin playing on the home stereo system. If a tapping interface were integrated into a children's toy, a child could communicate with the toy by tapping a certain song, such as 'Old Macdonald Had a Farm', causing the toy to respond by playing back a lively recording of the same tune. If integrated on a mobile device, a user could tap the rhythm of a song on his/her cell phone and the phone's software could immediately download a ringtone that contains a similar rhythm.

Another intriguing possible application would be to use tapping as a way to identify a person, in a situation where security is not a major concern. We propose to call this application area 'Authentication by Tapping'. For example, imagine that everyone in a household has a personalized rhythm, which could be fairly short. A child could configure the door to his/her room to only unlock to his/her 'secret knock' (but the parents would naturally have an override feature).

8 Conclusion

Through illustrating our online music search system that utilizes Query by Tapping, we have argued that tapping is a highly promising paradigm for human-computer interaction. By creating a web-based Query by Tapping system, and making it freely accessible on the Internet, we have been able to expose our system to a broad spectrum of users. We have presented good evidence from our study to show that tapping is an intuitive and effective means of communication, that can be used by people with varying musical abilities. We have presented

two novel encoding algorithms for rhythm, the Rhythmic Contour String algorithm and the Phrase String algorithm, which are effective enough to merit use in future applications which utilize tapping as a means of user input, such as in music education, mobile devices, or home integration.

While our approach to rhythmic searching allows humans to use a very natural and intuitive input method, it enables the computer system to make decisions that humans might find difficult. Untrained humans can mimic or shadow the clapping of another human, and can clap out the rhythm of a song they already know, but sometimes have a hard time when asked to identify a song based on a monotone rhythm they hear. As future work, we propose an experiment whereby the computer system's skill at recognizing songs based on rhythm would be compared to humans. While such an experiment would be akin to a sort of limited Turing Test for rhythm (similar to a Feigenbaum test [7]), which in value is debatable [16], it might also be interesting to compare our system's performance against both expert musicians as well as untrained humans. In any case, the broad range of potential applications involving tapping as a means of human-computer interaction shall make Query by Tapping an attractive focus of future study.

Acknowledgements. Special thanks to the SFU School of Computing Science, the SFU Faculty of Applied Science, the SFU Faculty of Business Administration, the SFU Faculty of Arts, and the Simon Fraser Student Society for providing funding and support of this project, to allow us to present our demonstration at AAAI-05 in Pittsburgh in July 2005. Also thanks to series editor Yang Cai from Carnegie Mellon University for his interest in our work.

References

[1] Aarts, E.: Ambient Intelligence: A Multimedia Perspective. Multimedia, IEEE. **11, 1** (2004) 12–19

[2] Bainbridge, D., Nevill-Manning, C., Witten, I., Smith, L., and McNab, R. Towards a Digital Library of Popular Music. Proceedings of the Fourth ACM Conference on Digital Libraries. ACM Press: NY. (1999)

[3] Chklovski, T., and Gil, Y.: Towards Managing Knowledge Collection from Volunteer Contributors. Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05). (2005)

[4] Chklovski, T.: 1001 Paraphrases: Incenting Responsible Contributions in Collecting Paraphrases from Volunteers. Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05). (2005)

[5] Doraisamy S., and Ruger, S.: An Approach Towards a Polyphonic Music Retrieval System. Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001. (2001)

[6] Eisenberg, G., Batke, J-M., and Sikora, T.: BeatBank - An MPEG-7 Compliant Query by Tapping System. 116th AES Convention, Berlin. (2004)

[7] Feigenbaum, E.: Some Challenges and Grand Challenges for Computational Intelligence. Journal of the ACM (JACM). **50, 1** (2003) 32–40

[8] Ghias, A., Logan, J., and Chamberlin, D.: Query by Humming - Musical Information Retrieval in an Audio Database. ACM Mulitmedia 95 - Electronic Proceedings. (1995)
[9] Gomez, E., Klapuri, A., Meudic, B.: Melody Description and Extraction in the Context of Music Content Processing. Journal of New Music Research. Routledge. **32, 1** (2003) 23–40
[10] Haitsma, J., and Kalker, T.: A Highly Robust Audio Fingerprinting System. International Symposium on Musical Information Retrieval (ISMIR2002). (2002) 144–148
[11] Harb, H., and Chen, L.: A Query by Example Music Retrieval Algorithm. 4th European Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS03). (2003) 122–128
[12] Kline, R., and Glinert, E.: Approximate Matching Algorithms for Music Information Retrieval Using Vocal Input. Proceedings of the Eleventh ACM International Conference on Multimedia. ACM Press: NY. (2003)
[13] Kosugi, N., Nishihara, Y., Sakata, T., Yamamuro, M., and Kushima, K.: A Practical Query-by-Humming System for a Large Music Database. Proceedings of the Eighth ACM International Conference on Multimedia. (2000) 333–342
[14] Kosugi, N., Nagata, H., and Nakanishi, T.: Query-by-Humming on Internet. Lecture Notes in Computer Science. Springer-Verlag. **2736** (2003) 589–600
[15] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. (Feb. 1966) 707–710.
[16] Luger, G.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Addison-Wesley. **5th ed.** (2005) Ch 1
[17] Melucci, M., and Orio, N.: Musical Information Retrieval Using Melodic Surface. Proceedings of the Fourth ACM conference on Digital Libraries. (1999) 152–160
[18] The Open Music Encyclopedia. Online Resource. `http://www.musicpedia.com` (2005)
[19] Parsons, D.: The Directory of Tunes and Musical Themes. Spencer Brown. (1975)
[20] Peters, G., Anthony, C., and Schwartz, M.: Song Search and Retrieval by Tapping. Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05). (2005) 1696–1697
[21] Peters, G., Anthony, C., and Schwartz, M.: The Song Tapper. Online Resource. `http://www.songtapper.com` (2005)
[22] Phillips-Silver, J., and Trainor, L.: Feeling the Beat: Movement Influences Infant Rhythm Perception. Science. **308** (2005) 1430–1430
[23] Jyh-Shing, R., Hong-Ru, L., and Chia-Hui, Y: Query by Tapping: A New Paradigm for Content-Based Music Retrieval from Acoustic Input. Lecture Notes In Computer Science. **2195** (2001) 590–597
[24] Shmulevich, I., Yli-Harja, O., Coyle, E., Povel, D.-J., and Lemstrom, K.: Perceptual Issues in Music Pattern Recognition Complexity of Rhythm and Key Finding. Proceedings of the AISB 99 Symposium on Musical Creativity. (1999) 64–69
[25] Sun, W., and Manber, U.: Fast Text Searching: Allowing Errors. Communication of the ACM. **35(10)** (1992) 83–91
[26] Tseng, Y.: Content-based Retrieval for Music Collections. Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM Press: NY. (1999)
[27] Uitdenbogerd, A., and Zobel, J.: Manipulation of Music For Melody Matching. ACM Multimedia 98. (1998) 235–240

[28] Von Ahn, L., and Dabbish, L.: Labeling Images with a Computer Game. ACM CHI. (2004)
[29] Wang, Y., Kan, M., Nwe, T., Shenoy, A., and Yin, J.: LyricAlly: Automatic Synchronization of Acoustic Musical Signals and Textual Lyrics. Proceedings of the 12th Annual ACM International Conference on Multimedia. ACM Press: NY. (2004)